

Deploy Your Django + React.js app to Heroku

#django #react #javascript #python



Shakib Hossain Jul 26, 2019 • Updated on Sep 28, 2019 • 8 min read

Nowadays, in most cases we see that there's a backend that provides an API and various front-end technologies like React, Vue, Mobile apps use this API endpoints to provide a user interface to the user. This method of development is becoming more and more popular with the rise in popularity of the great JavaScript frameworks like React, Vue, Angular etc.

There are mainly two ways you can deploy this kind of web apps:

- Separating Backend and Frontend: In this method, you server your back-end and
 front-end apps separately and they connect to each other with their respective URIs.
 One major overead of this approach is you have to configure cors yourself. If you
 don't know about cors you can learn more here.
- Serving from the same host: In this method you will be serving the app from the same URI so it removes the cors overhead. Also, it makes it easier to maintain smaller-medium sized apps. You don't want to create two separate repositories for

 \Diamond

35

111

Overview

I will show you how I integrated my **Django** app with my **React.js** front-end. We will follow the below steps:

- Generate a React app with create-react-app
- Create virtualenv and install necessary dependencies
- Generate a django project inside our React app
- Update both app settings
- Deploy to Heroku

The code example shown in this tutorial is available here.

Setup

I am listing the tech stack I am using below, just in case:

- Node 11.15.0
- Python 3.7.3
- yarn 1.17.3 (Node package manager)
- poetry 0.12.16 (Python package manager)

P.S. <u>poetry</u> is fairly new to the Python community. This makes the dependency management of python projects much more convenient. Also, similar to pipenv this handles the virtualenv for you. You can use this one or just use pip, pipenv or any other solution you like.

Generating React App

First, We have to generate our react application which will work as our front-end. For this tutorial, I'll name the project <code>django-react-boilerplate</code>. Change it to your liking. Let's create our react app. Here, I am generating a react app that uses **TypeScript**. You can ignore that by just ommitting the <code>--typescript</code> part from the above command. Run the below command to generate your React app:

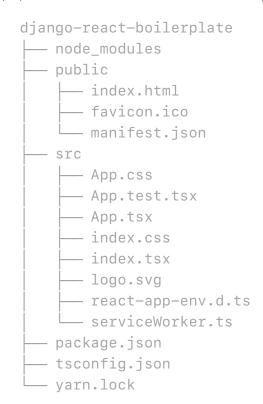
\$ yarn create react-app django-react-boilerplate --typescript

 \Diamond

35

111

52



Create Python Virtualenv

I will use **Poetry** to do this. So, if you are following exactly step-by-step you have to install poetry. You can get instructions from here. You are free to choose any other virtual environment solution you like in your case. First of all let's change directory to the generated react app. And then initialize poetry. It will ask you some general project related questions. You can choose to answer them, otherwise default values from Poetry will be used. You can also install your dependencies when you are installing your app but I will not do that in my case. After following the above instructions your shell might look something like this.

```
$ cd django-react-boilerplate
$ poetry init

This command will guide you through creating your pyproject.toml config.

Package name [django-react-boilerplate]:

Version [0.1.0]:

Description []:

Author [Shakib Hossain <shakib609@gmail.com>, n to skip]:

License []:

Compatible Duthon versions [A2 7]:

$\infty$ 8 $\infty$ 52 ***
```

```
Would you like to define your dev dependencies (require-dev) interactively (yes/
Generated file

[tool.poetry]
...
Do you confirm generation? (yes/no) [yes] yes
```

After finishing up generating your pyproject.toml file. You can now move on to installing the dependencies which we will need for our project. Let's install them:

```
$ poetry add django djangorestframework whitenoise gunicorn django-heroku
```

The above command will generate a virtualenv for you and install all the dependencies into it.

P.S. You might face problems while installing django-heroku if you don't have postgresql installed.

Generate Django App

Now it's time to generate our django app. We have to first enable our virtualenv. If you're using poetry then follow along, otherwise use your solutions method of activating the virtualenv. **Poetry** users can activate their virtualenv using the below command:

```
$ poetry shell
```

After activating the shell now we have access to our django python package and scripts that come with that package like django-admin. Let's generate our project inside the django-react-boilerplate directory. I am naming my backend project name backend. You're free to choose your own. Run the below command to generate the project inside the current directory:

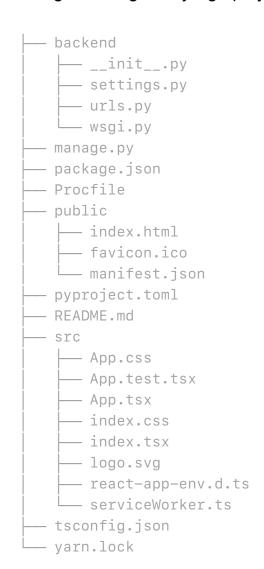


35



52

After generating the django project our project structure will look something similar to this.



Update Settings

First change that we will do is add a proxy key to our package.json. This will proxy all our API requests in development. You can learn more about it here. Add the following line to your package.json file.

```
"proxy": "http://localhost:8000"
}
```

After that, we have to create a directory named static inside the public directory. We will move the contents of the public directory into this new static directory except the

 \bigcirc

35

8

52

We have to move these files, so that when we build our React app by executing yarn build we will get these files inside a build/static directory, which we will use as our Django projects STATIC_ROOT.

Now, according our directory structure we have to refactor the public/index.html file. Open public/index.html file and update the favicon.ico and manifest.json urls to /static/favicon.ico and /static/manifest.json.

All the configuration to our React app is done. Now, we have to configure our Django project.

We mainly have one HTML file to serve(the React app generated HTML file). Let's create a view in our django app to serve this HTML file. I'll use Generic TemplateView to create the view. Create a views.py file inside the backend directory and add the below python code to the file:

```
from django.views.generic import TemplateView
from django.views.decorators.cache import never_cache

# Serve Single Page Application
index = never_cache(TemplateView.as_view(template_name='index.html'))
```

One thing to notice here that, I am using the <code>never_cache</code> decorator while initializing the <code>index</code> view. This decorator is pretty straight-forward. This adds headers to a response so that it will never be cached. We will be generating our <code>index.html</code> file from our React app which might change any time. That's why we do not want any browser to cache obsolete <code>index.html</code> file.

We've wrote the index view. Now let's add it to the urls.py. We will serve the index.html from our root url. Now open your urls.py and update it according to the code below:

 \Diamond

35

111

52

```
from .views import index

urlpatterns = [
    path('', index, name='index'),
    path('admin/', admin.site.urls),
]
```

Most of our work is done. All now we have to do is update our backend/settings.py file. Here, we'll first do everything as instructed in django-heroku documentation. After applying these changes, Our app won't work straightaway. We have to update our settings.py file further to make it work. First, add whitenoise and rest_framework to your INSTALLED_APPS like below. You have to list whitenoise right before django.contrib.staticfiles. And we also have to add the whitenoise middleware right after Djangos SecurityMiddleware.

Now, we have to update our TEMPLATES settings, so that our django app can find the index.html we referred to in our backend/views.py file. You can add additional directories you want to include here too.

52

}

Almost ready! We have to update our STATIC file related settings and move them to the bottom of the backend/settings.py file. Update your settings.py file like below:

```
# Import django_heroku module at the top of your settings.py file
import django_heroku
...
...

# Configure app for Heroku deployment
django_heroku.settings(locals())

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/
STATIC_URL = '/static/'
# Place static in the same location as webpack build files
STATIC_ROOT = os.path.join(BASE_DIR, 'build', 'static')
STATICFILES_DIRS = []

# If you want to serve user uploaded files add these settings
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'build', 'media')

STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

Testing Our Setup

We are now ready to test our app. A few things we have to keep in mind in this setup:

- Always run yarn build after you've updated your front-end
- While developing, you have to run the react server and django server separately to make use of the built-in hot-reload of these servers.

Now, run the below commands to test whether our app is serving the files correctly.

Open your preffered browser and navigate to localhost:8000. You should see the default React app.

Preparing for Heroku Deployment

First, let's create our heroku app with the below command(Make sure you have herokucli installed):

```
$ heroku create drt-boilerplate
```

Add nodejs and python buildpacks and the postgresql addon to our app.

```
$ heroku buildpacks:add --index 1 heroku/nodejs
$ heroku buildpacks:add --index 2 heroku/python
$ heroku addons:create heroku-postgresql:hobby-dev
```

Create a Procfile:

```
release: python manage.py migrate web: gunicorn backend.wsgi --log-file -
```

Here, the release option makes sure to run your django migrations after each deploy. And the web option serves your django application using gunicorn HTTP server.

You have to generate a requirements.txt file for heroku deployments. So, don't forget to do that.

```
$ poetry run pip freeze > requirements.txt
```

We are ready to push the first version of our app. Create a git repository and make a commit. After that, push the files to heroku by running the below command:

```
$ git push heroku master
```

35

This will trigger a deploy and show you your deploy progress. After a successful deploy it





52

This article was first published <u>here</u>.

Discussion (15)

Subscribe



Add to the discussion



Ryuichi Miyazaki • Feb 29 '20

Another thing to add for those using this tutorial as a skeleton for their project. Once you update the Frontend you will need to run two commands:

- 1. npm run build
- 2. python manage.py collectstatic

Otherwise you will run into a Template not found error.

Happy Hacking!







Ryuichi Miyazaki • Feb 20 '20 • Edited

X

[SOMEWHAT SOLVED] Hi Shakib!

Great post! It was exactly what we were looking for given our tech stack.

Unfortunately I am having issues with building the project and I'm not sure what I'm doing wrong.

I get this error:

remote: ----> Build succeeded!

remote: ! Unmet dependencies don't fail yarn install but may cause runtime issues

remote: github.com/npm/npm/issues/7494

remote:

remote: ----> App not compatible with buildpack: <u>buildpack-registry.s3.amazonaws.co...</u>

remote: More info: devcenter.heroku.com/articles/buil...

remote:

remote: ! Push failed



35



52

remote: ! Push rejected to r-d-test.

remote:

If anyone can give any hints to what I might be doing wrong that would be great.

I followed the directions down to the key and checked the current comment suggestions as well...

The Procfile and requirements.txt are correctly in the root of the project as well...

It looks like my react part of the project works fine but something wrong with the python buildpack...





Ryuichi Miyazaki • Feb 24 '20 • Edited

[UPDATE]

So I discovered that we needed to update the .wsgi file which I found in the source code but clearly not a part of the tutorial. I think it would be helpful to add that section after the testing of the app if it works locally.

[EDIT]

Sorry, so when I ran the deployment when I did this I did not realize my partner had changed it.

The original settings should work.

♥ 2 Q



Joe Szaf • Oct 1 '20

Ran into the same problem.

It was fixed when I actually committed my work before pushing to master.

Specifically, I ran the following before "\$ git push heroku master":

git add . git commit -m "initial commit"

35

111

52



Shakib Hossain 🚡 • Mar 9 '20

Sorry for the late reply. Can you share with me what modifications you had to make to the wsgi.py file? In my case I didn't have to modify it, that's why I did not include it in the tutorial.





Ryuichi Miyazaki • Apr 24 '20

Sorry for the late reply.

I had finished the project and then COVID happened.

I looked back and actually the default wsgi.py settings were correct. My partner had changed it without telling me...

♥ 2 Q



Lukas Gerhardt • Mar 27 '20

Hey Ryuichi,

I ran into the same error you were having. Can you please share what fixed the problems with the python buildpack in the wsgi.py file?

♥ 1 Q



so2liu • Feb 12 '20

Hi there, thank you for your guide. I have followed to the end and met a problem that build success but deployment failed.

success but deployment failed.

psycopg2.OperationalError: FATAL: remaining connection slots are reserved for nor

The above exception was the direct cause of the following exception:

. . .

django.db.utils.OperationalError: FATAL: remaining connection slots are reserved

 \Diamond

35

5

52

Stackflow said this is something about connection number. But it's a new app and should not have several connections. Any clue? Thank you very much!

♥ 1 Q



so2liu • Feb 12 '20

I solved this problem by deleting this line in Procfile:

release: python manage.py migrate

Following this stackflow Link.

Of course I don't know what happened. I would appreciate it if you explain somehow.

♥ 1 Q



Mohammed Swalih • Jun 19 '20

Hi folks, I am getting this error with blank page whiled deployed on heroku, Uncaught SyntaxError: Unexpected token '<', though i am able to run the app on local host..could any one suggest ..:) thanks in advance

 \bigcirc 1 \bigcirc



Adam • Sep 28 '19

Thank you for this guide and repo! It was very helpful.

At first it didn't work for me, but adding this on line 12 in settings.py fixed it: import django_heroku

♥ 3



Shakib Hossain 💍 • Sep 28 '19

Glad that you found the post useful and thanks for pointing out the mistake. I forgot to import that when I was writing the tutorial.

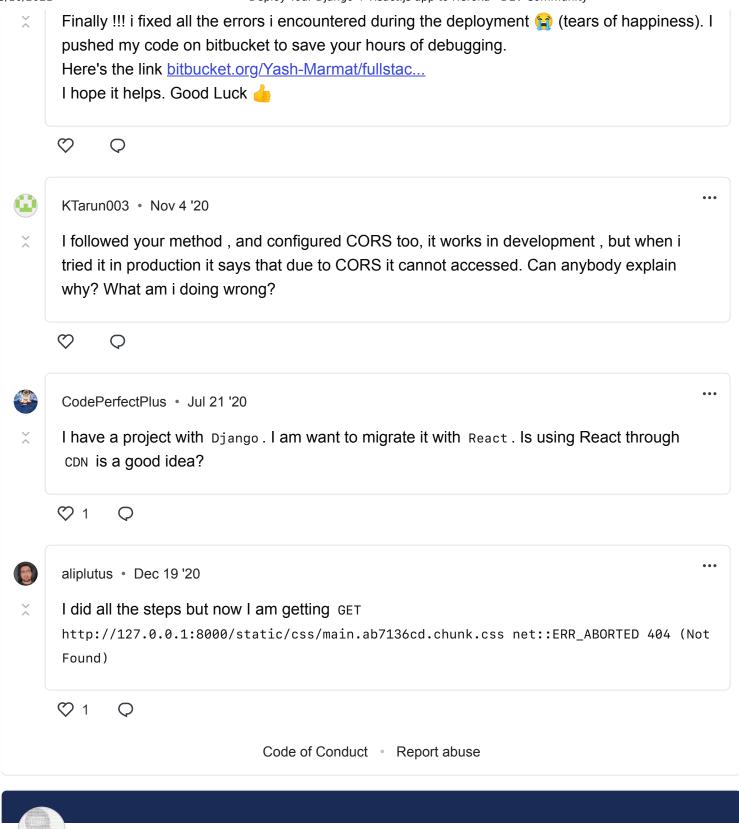
♥ 2 <</p>

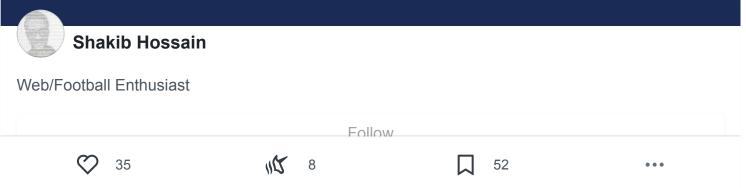
35

 \Diamond

1st

52





Software Developer at Goama Games

LOCATION

Chittagong

EDUCATION

B.Sc in Computer Science & Engineering

JOINED

Feb 19, 2019

Trending on DEV Community



My Journey into Developer Relations

#devrel #career #nrwl #opensource

Creating beautiful command-line interfaces in Python

#python #programming #tutorial

The ultimate guide for data structures & algorithm interviews 🔥

#career #beginners #algorithms #javascript

 \heartsuit

35



52